Seamless Mobile Computing on Fixed Infrastructure

M. Kozuch, M. Satyanarayanan, T. Bressoud, C. Helfrich, S. Sinnamohideen

IRP-TR-03-11

Research at Intel

Seamless Mobile Computing on Fixed Infrastructure

Michael Kozuch[‡], M. Satyanarayanan[†‡], Thomas Bressoud[‡⋆], Casey Helfrich[‡], Shafeeq Sinnamohideen[†‡]
[‡]Intel Research Pittsburgh, [†]Carnegie Mellon University, and [⋆]Denison University

1 Introduction

The term "mobile computing" typically evokes an image of a user with a laptop, handheld or wearable computer. But pervasive computing infrastructure can virtually eliminate the need to carry hardware. In this paper, we present an approach to mobile computing that is becoming increasingly viable as Internet connections and PCs proliferate.

The plummeting cost of computing hardware makes it possible for us to imagine a world in which coffee shops, airport lounges, dental and medical offices, and other semi-public spaces provide free network-connected hardware for their clientele. Even the fold-out tray at every seat in an aircraft or commuter train could be a laptop for use by its current passenger. Such hardware may help users make productive use of unanticipated slivers of time at various locales.

Our vision is to make personal computing a commodity from a hardware perspective, yet fully retain customization from a user's perspective. This vision is crisply captured by the slogan "Any computer can be *your* computer." In other words, only when a user sits down to use a computer does it magically acquire the customizations and unique state that is specific to that user. Like light at the flip of a switch, or water at the turn of a faucet, personal computing then becomes available on demand at any location. Only on rare occasions will a user need to carry mobile computing hardware.

Three important criteria must be met for this vision to materialize. First, a user should not have to waste time customizing each use of a machine. When he starts, his personalized computing environment should appear almost immediately. When he is done, it should just take a mouse click to save his session. Only then can the user benefit from brief usage periods at different machines.

Second, the hidden costs of owning and deploying machines must be low. From the viewpoint of a coffee shop owner or a doctor's office, the infrastructure should require no more management or system administration than a chair, table lamp or fold-out tray. Turning off, moving, adding or removing machines should be trivial actions that require no central coordination. When a user departs, he should leave behind no permanent state on a machine.

Third, a user has to be confident that a machine is safe to use. He needs to be sure that a previous user or an unscrupulous employee on the premises has not left behind a Trojan horse. In turn, the machine may need to be confident of the identity of the user.

This paper addresses the first two criteria. We do not address the difficult problem of establishing trust in hardware. We are relying on the work of other researchers [8, 11] to develop workable solutions for this problem. Until then, we assume that the physical security of deployed machines is assured by their owners. The problem of authenticating users to machines is a much simpler one, with off-the-shelf solutions such as Kerberos.

We describe a mechanism called *Internet Suspend/Resume (ISR)* that rapidly personalizes and depersonalizes anonymous hardware for transient use. As its name implies, ISR mimics the closing and opening of a laptop. A user can suspend work on one machine, travel to another location, and resume work on another machine there. The user-visible state at resume is exactly what it was at suspend. ISR enables a form of mobile computing in which a user carries no hardware, yet sees functionality and performance as if carrying a laptop.

We have implemented ISR on Linux by layering virtual machines (VMs) on a location-transparent distributed file system that aggressively caches data. Each VM encapsulates execution and user customization state. The distributed file system transports that state across space (from suspend site to resume site) and time (from suspend instant to resume instant). Section 3 explains why virtualization and caching are the keys to precise customization and simple administration.

2 Background

ISR's thick-client approach to mobility is fundamentally different from thin-client approaches such as Infopad [10], SLIM [7], VNC [3] and X-Move [9]. ISR offers crisp and stable interactive performance under conditions of high network latency, network congestion or even network failure. In contrast, interactive performance can be very frustrating for a thin client user under any of these conditions. The building blocks of modern-day user interfaces such as scrolling, highlighting and pop-up menus all assume very low latency between the GUI and its applications. These mechanisms fail to provide a satisfactory user experience when latency is high or uneven.

Chen and Noble [1] were the first to observe that the clean encapsulation of state by a VM suggests the possibility of simple state migration. The first experimental evidence confirming Chen and Noble's hypothesis was reported by us in mid-2002 [2]. Our implementation used a very simple copyout/copyin migration technique that was only adequate

as a proof of concept. Recently, Sapuntzakis et al. [4] have reported on a number of optimizations that yield impressive reductions in the volume of data transferred during VM migration. Such optimizations complement our work.

In the rest of this section, we provide a brief VM tutorial to define the terminology used in the paper. A VM is a software emulator with excellent fidelity; neither system nor application software executing within the VM can tell that it is not directly executing on bare hardware. Emulation is performed by a *virtual machine monitor* (VMM). We use VMware Workstation (abbreviated to just "VMware"), a commercial VMM for the Intel x86 architecture. VMware operates in conjunction with a *host* operating system, relying on it for services such as device management. Figure 1 illustrates this situation. VMware runs on several host OSes and supports a wide range of guest OSes including Windows 95/98, Windows 2000/XP, and Linux.

VMware maps the state of each supported VM to host files. For a VM named `testvm`, the hardware configuration is described in the file `testvm.cfg`. Each of its virtual disks corresponds to a file: `testvm1.vmdk`, `testvm2.vmdk`, and so on. When a VM is suspended, its volatile state is saved in the file `testvm.vmss`. After suspension, `testvm`'s files can be copied to another machine with a similar hardware architecture. VMware on that machine can then continue execution of `testvm`. In other words, `testvm` has been migrated from one machine to another.

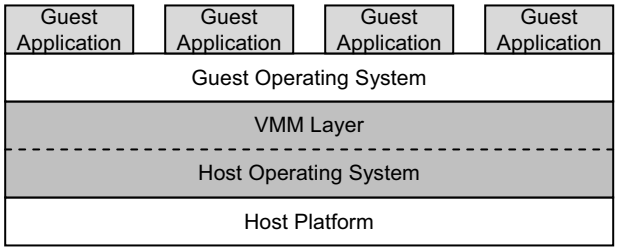

Figure 1. Virtual Machine Layered on Host OS

3 Design Rationale

3.1 Accurate Personalization

Simplicity was the driving force behind our decision to use a VM-based approach for state encapsulation. This approach eliminates the need for modifications to guest applications or guest operating systems. In particular, ISR supports unmodified Microsoft software such as any of the Windows operating systems and the Office application suite. The average user can thus benefit immediately from ISR.

A VM-based system is easier to deploy than a design based on process migration. In contrast to the well-known difficulties of process migration, VM migration is simpler because volatile execution state is much better encapsulated. It is also tolerant of greater disparity between the source and target systems across which migration occurs. For process migration to succeed, there has to be a very close match between host and target operating systems, language runtime systems, and so on. In contrast, VM migration only requires a compatible VMM at the target.

3.2 Easy Site Management

Once a VM-based approach was chosen, the next critical decision was how to store and transport VM state between suspend and subsequent resume. If the resume site is known with certainty at suspend, direct data transfer is the simplest and most efficient approach. However, there are many situations where a user may not resume for many hours or days. When he does resume, his location may be at a site other than originally anticipated. In the face of such uncertainty, direct data transfer would require the suspend site to preserve the user's VM file state for an extended period of time. Until resume, the suspend site cannot be turned off or unplugged from the network. This violates our goal of granting full autonomy to ISR sites to simplify their management.

These considerations lead to a design in which the Internet is the true home of VM state, and ISR sites are merely temporary usage points of that state. Since VMware stores state in files, a distributed file system is the obvious form of Internet storage for ISR. Distributed file systems are a mature technology, with designs such as AFS that aggressively cache data at clients for performance and scalability.

The use of such a distributed file system, with all ISR sites configured as clients, is the key to mobility. Demand caching at a resume site ensures that relevant parts of VM state follow a user from suspend to resume. The use of a distributed file system also simplifies management of ISR sites. Since an ISR site holds no user-specific state except during active use, it can be treated like an appliance. An idle site can be turned off, moved, or discarded at will without centralized coordination or notification. Thus, although data is transferred in two hops (suspend site to server, then server to resume site), the classic client-server model is a better match for ISR's design goals than a design in which ISR sites directly transfer data to each other.

The highly asymmetric separation of concerns made possible by a distributed file system reduces the skill level needed to manage ISR sites. Little skill is needed to maintain machines or to deploy new ones. System administration tasks that require expertise (such as backup, restoration, load balancing, and addition of new users) are concentrated on a few file servers administered by a small professional staff. We expect that server hardware and the professional staff to administer them will often be dedicated to a specific organization such as a company, university or ISP. Since locations such as coffee shops and doctors' offices are likely to be visited by ISR users belonging to many different organizations, domain-bridging mechanisms such as AFS *cells* or Kerberos *realms* will be valuable.

3.3 Large VM State

A key obstacle to using ISR is *high resume latency.* The state of a typical VM today is quite large: at least a few GB, and possibly many tens of GB. Naive approaches to transferring this state will result in intolerable resume latencies. Fortunately, a number of real-world usage considerations can mitigate these delays.

First, temporal locality is often present in the mobility patterns of users. For example, a common usage pattern we envision for ISR is a user working at home, suspending, traveling to his office, and resuming there; later in the day, he suspends at the office, returns home, and resumes. As another example, a worker in a corporate campus or a supervisor in a factory might visit the locations of his coworkers many times in the course of a day, and may resume his VM at any of those locations. If VM state is cached at fine granularity, this temporal locality in physical mobility will translate into locality of file references. With large enough file caches at ISR sites, resume latency will be much reduced since misses will occur only on state that has changed since the last visit to that location.

Second, it is sometimes possible to confidently predict where a user will resume work. With the assistance of higher-level software, it may be possible to identify likely resume sites and to proactively transfer state to those sites. This will lower resume latency. Since proactivity merely requires warming a file cache in our design, the consequences of acting on an erroneous prediction are mild. Useful file cache state may be evicted and network bandwidth may be wasted, but there is no loss of correctness or need for cleanup actions.

Third, although VM state is enormous, much of it consists of disk state that rarely changes after initialization. For example, installing Windows XP and the Microsoft Office suite on a small VM configuration can consume one-quarter to one-half its virtual disk capacity. Since this state is identical on other similarly-configured VMs, it could be captured on read-only media such as CDROMs and distributed to ISR sites that have poor connectivity to the Internet. At resume, ISR software could synthesize large parts of the VM state from the read-only media rather than demand-fetching it over a slow network. The risk is, of course, that the read-only media may be out of date with respect to VM upgrades reflected on the file server. A workable solution must allow the use of read-only media without compromising the consistency of VM state.

3.4 Transportable Storage

Today, USB and Firewire storage devices in the form of small, unobtrusive storage keychains or microdrives are widely available. For brevity, we refer to such transportable storage devices as *dongles* in this paper. Today, the smallest dongles have capacities up to 1GB, while larger dongles have capacities up to a few tens of GB. A dongle could reduce resume latency at poorly-connected ISR sites by acting as a local source of critical data. Users may be willing to carry a dongle if the benefits are appreciable.

There are at least two risks with using a dongle. First, it could be out of date with respect to current VM state in the distributed file system. This might happen, for example, if the user forgets to update the dongle at suspend or if he absent-mindedly picks up the wrong dongle for travel. Second, the dongle may be lost, broken or stolen while travelling. These considerations suggest that ISR should treat a dongle only as a performance enhancement, not as a substitute, for the underlying distributed file system. They also suggest that data on dongles be self-validating — a stale dongle may not improve performance, but should do no harm.

Our solution is a new mechanism called *lookaside cache miss handling (LKA)* that enables a client to obtain file data content from a source other than a file server. LKA enables transportable storage devices and distribution media to be integrated into a distributed file system without compromising consistency or security, or complicating system aministration.

4 Architecture and Implementation

4.1 Distributed File System

We had a choice of three distributed file systems for ISR: NFS, AFS and Coda[6]. Although NFS is the most widely supported of these, it only caches file blocks in memory; it does not cache data persistently in the local file system. Hence, the cache size at an ISR site can be no larger than its memory size, which is typically much smaller than total VM state size. This limits ISR's ability to take advantage of temporal locality of access to VM state. Further, since NFS tends to perform poorly in WAN environments, its use would limit our ability to support ISR anywhere on the Internet.

Both AFS and Coda clients cache files on their local disks, and perform acceptably in WAN environments. We decided to use Coda for a number of reasons. First, Coda's support for hoarding (anticipatory cache warming) provides a clean interface to exploit advance knowledge of resume site. Second, Coda supports trickle reintegration. This propagates dirty client state to file servers in the background from poorly-connected ISR sites and thus can reduce suspend latency. Third, the user-space (rather than in-kernel) implementation of the Coda client simplifies our LKA extensions.

The Coda security model assumes a small trusted pool of servers and a large untrusted collection of clients. When a user walks up to an ISR site, he must explicitly authenticate himself via Kerberos before he can resume. It is up to the user to ensure that an ISR site is safe to use.

4.2 Data Layout

We represent very large VMware files as a directory tree rather than a single file in Coda. Virtual disk state is divided

into 256KB chunks, and each chunk is mapped to a separate Coda file. These files are organized as a two-level directory tree to allow efficient lookup and access of each chunk. Our choice of 256KB is based on a trace-driven analysis of chunk size on performance. The other VM state files mentioned in Section 2 are also stored in Coda. The large memory state file (`testvm.vmss`) is stored in compressed form and is uncompressed into `/tmp` just prior to resume.

4.3 Client Architecture

Figure 2 shows the client architecture that we have developed to interface VMware to Coda. A loadable kernel module called *Fauxide* serves as the device driver for a pseudo-device named `/dev/hdk` in Linux. A VM is configured to use this pseudo-device as its sole virtual disk in "raw" mode. Disk I/O requests to `/dev/hdk` are redirected by Fauxide to a user-level process called *Vulpes*. It is Vulpes that implements VM state transfer policy, as well as the mapping of VM state to files in Coda. Vulpes also controls the hoarding of those files and their encryption. Since Vulpes is outside the kernel and fully under our control, it is easy to experiment with a wide range of state transfer policies.

4.4 Lookaside Cache Miss Handling

As mentioned in Section 3.4, we have extended Coda with a simple yet versatile mechanism called LKA to take advantage of dongles and read-only distribution media. LKA consists of three parts: a small change to the client-server protocol to obtain the SHA-1 hash of a file as part of its status; a quick index check (the "lookaside") in the code path for handling a cache miss; and a tool for generating LKA indexes. Dynamic inclusion or exclusion of one or more indexes is done through user-level commands on a Coda client.

The lookaside occurs between the RPCs to fetch file status and file contents. Since the client possesses the hash of the file at this point, it can cheaply consult one or more local LKA indexes to see if a local file with identical SHA-1 value exists. Trusting in the collision resistance of SHA-1, a copy of the local file can then be a substitute for the RPC to fetch file contents. To detect version skew between the local file and its index, the SHA-1 hash of the local file is recomputed. In case of a mismatch, the local file substitution is suppressed and the cache miss is serviced by contacting the file server. Coda's consistency model is not compromised, although some small amount amount of work is wasted on the lookaside path. Coda's callback mechanism ensures that cached hash information tracks server updates.

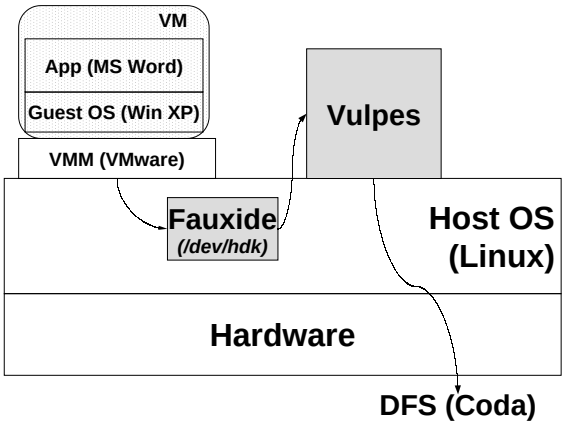

Figure 2. ISR Host Architecture

Lookaside can be directed to a local file tree, to a mounted storage device, or even to nearby NFS or Samba server. For a removable medium such as a dongle, the LKA index is located on the medium itself. This yields a self-describing storage device that can be used at any ISR site.

5 State Transfer Policies

Copyout/copyin is the most conservative endpoint in a spectrum of VM state transfer policies. All state is copied out at suspend; resume is blocked until the entire state has arrived. Three steps can be taken to shorten resume latency:

- Propagate dirty state to servers before suspend.
- Warm the file cache at the resume site.
- Allow resume to occur before full state has arrived.

These steps are not mutually exclusive, and can be combined in many different ways to generate a wide range of policies. We explore some of these policies below.

5.1 Characterizing Policies

The conceptual timeline shown in Figure 3 provides a uniform framework for our discussion of policies. The figure depicts a user initially working for duration *t1* at Internet location *site1*. He then suspends, and travels to Internet location *site2*. In some situations, the identity of *site2* is known (or can be guessed) *a priori*. In other situations, it becomes apparent only when the user unexpectedly shows up and initiates resume. The transfer of dirty state from *site1* to file servers continues after suspend for duration *t2*. There is then a period *t3* available for proactive file cache warming at *site2*, if known. By the end of *t3*, the user has arrived at *site2* and initiates resume. He experiences resume latency *t4* before he is able to begin work again. He continues working at *site2* for duration *t5* until he suspends again, and the above cycle repeats itself. With some state transfer policies, the user may experience slowdown during the early part of *t5* because some operations block while waiting for missing state to be transferred.

Note that Figure 3 is only a canonical representation of the ISR timeline. Many special or degenerate cases are possible. For example, *t2* may not end before resume if travel duration is very short. In that case, the residue of *t2* may add to *t4* in contributing to resume latency. On the other hand,

a clever state transfer policy may allow this residue to overlap *t4*. In other words, propagation of dirty state from the suspend site to file servers could overlap state propagation from those servers to the resume site. Another special case is when *t5* is very brief. With such a short dwell time, full VM state may never accumulate at *site2* — only enough to allow the user a few moments of work past the suspend point at the end of *t1*. While many such special cases are conceivable, the timeline in Figure 3 is likely to cover a wide range of common real-world scenarios.

5.2 Performance Metrics

From a user's viewpoint, the performance of ISR is defined by the answers to two questions:

- **Resume latency**: *How soon after I resume at a new site can I begin useful work?* The answer to this question corresponds to the period *t4*.
- **Slowdown**: *How much is my work slowed down after I resume?* The answer corresponds to the performance degradation during *t5*.

Ideally one would like zero resume latency and zero slowdown. In practice, there are trade-offs between the two. Policies that shrink resume latency may increase average slowdown and vice versa. Our goal is to quantify these trade-offs for typical ISR scenarios.

5.3 Policies Examined

5.3.1 Baseline

The baseline policy is a worst-case strawman. After suspend, all dirty state is transferred to the server during *t2*. The period *t3* is empty. Following resume, the entire VM state is transferred to the resume site during *t4* and pinned in the client cache. Note here that no state transfer occurs during either execution period *t1* or *t5*. This optimizes for execution speed at the cost of suspend and resume latency.

5.3.2 Fully Proactive

If we can predict *site2*, we can define a much more aggressive state transfer policy. At *site2*, this policy shifts the entire state transfer time from *t4* to earlier periods in the ISR timeline. During *t3* (or earlier, for any state already available at the servers) *site2* transfers all updated state to its local cache. Note that this includes both VM disk and memory state. At resume, all that remains is to launch the VM. We expect this policy to be most effective when a user is working among a small set of sites, such as when alternating between home and work.

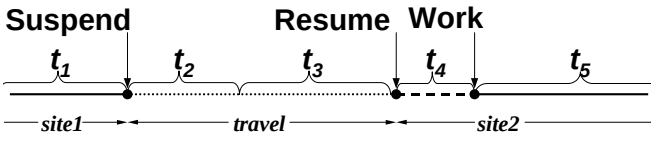

Figure 3. Conceptual ISR Timeline

5.3.3 Pure Demand-Fetch

Suppose a user arrives unexpectedly at an ISR site. If we wish to keep *t4* as short as possible, we can use a pure demand-fetch policy to amortize the cost of retrieving the VM disk state over *t5*. In this policy, only virtual memory state (the `testvm.vmss` file) is retrieved during *t4*; the transfer of disk state (the 256KB chunks corresponding to `testvm.vmdk` files) is deferred. As soon as the virtual memory state has arrived, the VM is launched. Then, during *t5*, disk accesses by the VM may result in Coda cache misses that cause slowdown.

5.3.4 Demand-Fetch with Lookaside

The performance of pure demand-fetch can be improved by using LKA in a number of different ways. If a user is willing to wait briefly at suspend, the `testvm.vmss` file and an LKA index for it can be written to his dongle. He can then remove the dongle and carry it with him. At the resume site, LKA can use the dongle to reduce *t4*.

If read-only or read-write media with partial VM state are available at the resume site, LKA can use them to reduce the cost of cache misses during *t5*. This will reduce slowdown after resume. When combined with the use of a dongle, this can improve both resume latency and slowdown.

6 Experimental Evaluation

6.1 Benchmark

ISR is intended for interactive workloads typical of laptop environments. We have developed a benchmark called the *Common Desktop Application (CDA)* that models an interactive Windows user. CDA uses Visual Basic scripting to drive Microsoft Office applications such as Word, Excel, Powerpoint, Access, and Internet Explorer. The operations mimic typical actions that might be performed by an office worker. CDA pauses between operations to emulate think time. The pause is typically 10 seconds, but is 1 second for a few quick-response operations.

6.2 Methodology

Our experimental infrastructure consists of 2.0 GHz Pentium 4 clients connected to a 1.2 GHz Pentium III Xeon server through 100 Mb/s Ethernet. All machines have 1 GB of RAM, and run RedHat 7.3 Linux. Clients use VMware Workstation 3.1 and have an 8 GB Coda file cache. The VM is configured to have 256 MB of RAM and 4GB of disk, and runs Windows XP as the guest OS. We use the NISTNet network emulator to control available bandwidth.

Without ISR support, the benchmark time on our experimental setup is 1071 seconds. In this configuration, the files used by VMware are on the local file system rather than on

/dev/hdk. The effects of Fauxide, Vulpes and Coda are thus completely eliminated, but the effect of VMware is included. The figure of 1071 seconds is a lower bound on the benchmark time achievable by any state transfer policy in our experiments.

6.3 Results: Baseline

Relative to the metrics described in Section 5.2, we expect the baseline policy to exhibit poor resume latency because all state transfer takes place during the resume step. We also expect network bandwidth to be the dominant factor in determining this quantity. The column of Figure 4 labelled "Baseline" confirms this intuition.

At 100 Mb/s, the resume latency is about 40 minutes. When bandwidth drops to 10 Mb/s, resume latency roughly doubles. The reason it does not increase by a factor of ten (to match the drop in bandwidth) is that the data transfer rate at 100 Mb/s is limited by Coda rather than by the network. Only below 10 Mb/s does the network become the limiting factor. The results in Figure 4 show that the baseline policy is only viable at LAN speeds, and even then only for a limited number of usage scenarios.

In contrast to resume latency, we expect slowdown to be negligible with the baseline policy because no ISR network accesses should be necessary once execution resumes. The "Baseline" column of Figure 5 confirms that slowdown is negligible at 100 Mb/s. The total running time of the benchmark increases from 1071 seconds to 1105 seconds. This translates to a slowdown of about 3.2%, where slowdown is defined as $(T_{\text{bw}} - T_{\text{noisr}})/T_{\text{noisr}}$, with T_{bw} being the benchmark running time at the given bandwidth and T_{noisr} its running time in VMware without ISR.

As bandwidth drops below 100 Mb/s, the "Baseline" column of Figure 5 shows that slowdown grows slightly. It is about 9.2% at 10 Mb/s, 18.8% at 1 Mb/s, and 31.6% at 100 Kb/s. This slight dependence on bandwidth is due to Coda background activity such as trickle reintegration.

6.4 Results: Fully Proactive

With a fully proactive policy one expects resume latency to be bandwidth-independent and very small because all necessary files are already cached. The "Fully Proactive" column of Figure 4 confirms this intuition. Resume latency is only 10 - 11 seconds at all bandwidths.

Post-resume ISR execution under a fully proactive policy is indistinguishable from the baseline policy. The user experience, including slowdown, is identical. Clearly, the fully proactive policy is very attractive from the viewpoint of resume latency and slowdown.

What is the minimum travel time for a fully proactive policy to be feasible? This duration corresponds to $t2 + t3$ in Figure 3. There are two extreme cases to consider. In the best case, the resume site is known well in advance and its cache has been closely tracking the cache state at the suspend site. All that needs to be transferred is the residual dirty state at suspend — the same state that is transferred to servers during *t2*. For our experimental configuration, we estimate this state to be about 47 MB at the mid-point of benchmark execution. Using observed throughput values in our prototype, this translates to minimum best case travel time of 45 seconds with a 100 Mb/s network, and about 90 seconds with a 10 Mb/s network. Both of these are credible bandwidths and minimum walking distances today between collaborating workers in a university campus, corporate campus or factory.

At lower bandwidths, we estimate the best case travel time to be at least 800 seconds (roughly 14 minutes) at 1 Mb/s, and 8000 seconds (roughly 2 hours and 15 minutes) at 100 Kb/s. The 14 minute travel time is shorter than many commutes between home and work, and bandwidths close to 1 Mb/s are available to many homes today. Over time, network infrastructure will improve, but travel times are unlikely to decrease.

In the worst case, the resume site has a completely cold cache and is only identified at the moment of suspend. In that case, $t3$ must be long enough to transfer the entire state of the VM. From the baseline resume latencies in Figure 4 and the value of $t2$ above, we estimate minimum travel time to be 2550 seconds (roughly 43 minutes) for a 100 Mb/s network, and 5250 seconds (88 minutes) for a 10 Mb/s network.

6.5 Results: Pure Demand-Fetch

In the pure demand-fetch policy, state transfer begins only at resume. However, in contrast to the baseline policy, only a very small amount of state is transferred. In our prototype, this corresponds to the compressed memory image of the VM at suspend (roughly 41 MB). The transfer time for this file is a lower bound on resume latency for pure demand-fetch at any bandwidth. As the "Pure Demand-Fetch" column of Figure 4 shows, resume latency rises from well under a minute at LAN speeds of 100 Mb/s and 10 Mb/s to well over an hour at 100 Kb/s.

We expect the slowdown for a pure demand-fetch policy to be very sensitive to workload. The "Pure Demand-Fetch" column of Figure 5 confirms this intuition. The total benchmark time rises from 1071 seconds without ISR to 1160 seconds at 100 Mb/s. This represents a slowdown of about 8.3%. As bandwidth drops, the slowdown rises to 30.1% at 10 Mb/s, 340.9% at 1 Mb/s, and well over an order of magnitude at 100 Kb/s. The slowdowns below 100 Mb/s will undoubtedly be noticeable to a user. But this must be balanced against the potential improvement in user productivity from being able to resume work anywhere, even from unexpected locations.

6.6 Results: Demand-Fetch with Lookaside

As discussed in Section 3.4, the use of transportable storage can reduce both the resume latency and slowdown of a

demand-fetch state transfer policy. Our experiments show that these reductions can be substantial.

The "Dongle LKA" column of Figure 4 presents our results for the case where a dongle is updated with the compressed virtual memory image at suspend, and used as a lookaside device at resume. Comparing the "Dongle LKA" and "Pure Demand-Fetch" columns of Figure 4 we see that the improvement is noticeable below 100 Mb/s, and is dramatic at 100 Kb/s. A resume time of just 12 seconds rather than 317 seconds (at 1 Mb/s) or 4301 seconds (at 100 Kb/s) can make a world of a difference to a user with a few minutes of time in a coffee shop or a waiting room.

To explore the impact of LKA on slowdown, we constructed a DVD with the VM state captured after installation of Windows XP and the Microsoft Office suite, but before any user-specific or benchmark-specific customizations. We used this DVD as a lookaside device for LKA during the running of the benchmark. The "DVD LKA" column of Figure 5 presents our results. Comparing the "DVD LKA" and "Pure Demand-Fetch" columns of Figure 5, we see that benchmark time is reduced at all bandwidths. The reduction is most noticeable at lower bandwidths.

6.7 Off-machine Lookaside

We have recently extended LKA to use off-machine *content-addressable storage (CAS)*. The growing popularity of planetary-scale services such as PlanetLab, distributed hash-table storage such as Pastry and Chord, and logistical storage such as the Internet Backplane Protocol, all suggest that CAS will become a widely-supported service in the future. For brevity, we refer to any network service that exports a CAS interface as a *jukebox*. When presented with a hash, the jukebox returns content matching that hash, or an error code indicating that it does not possess requested content.

The "Jukebox LKA" column of Figure 5 shows the performance benefit of using a LAN-attached jukebox with same contents as the DVD of Section 6.6. Comparing the "Jukebox LKA" and "DVD LKA" columns of Figure 5, we see that the improvement in the two cases is similar relative to the "Pure Demand-Fetch" column.

7 Related Work

Although ISR is new, it is only the latest step in a long historical evolution toward user mobility in fixed infrastructure. The earliest form of user mobility, dating back to the early 1960's, was supported by timesharing systems attached to "dumb" terminals. A user could walk up to any terminal and access his personal environment there. Thin clients are the modern-day realization of this capability, providing just enough compute power to support GUIs.

Thick client strategies became possible after the birth of personal computing circa 1980. The vision of walking up to any machine and using it as your own dates back at least to the mid-1980's. Both location transparency and client caching in AFS were motivated by this consideration. To quote a 1990 AFS paper [5]: "User mobility is supported: A user can walk up to any workstation and access any file in the shared name space. A user's workstation is 'personal' only in the sense that he owns it." This capability falls short of ISR in two ways. First, only persistent state is saved and restored; volatile state such as the size and placement of windows is not preserved. Second, the user sees the native operating system and application environment of the client; in many cases, this may not be his preferred environment.

ISR bears a close resemblance to *process migration*. The key difference lies in the level of abstraction at which the two mechanisms are implemented. ISR operates as a hardware-level abstraction, while process migration operates as an OS-level abstraction. In principle, this would seem to put ISR at a disadvantage because hardware state is much larger. In practice, the implementation complexity and software engineering concerns of process migration have proved to be greater challenges. Although successful implementations of process migration have been demonstrated, no OS in widespread use today supports it as a standard capability.

8 Conclusion

ISR is a mechanism that accurately encapsulates all the customizations that a typical user cares about, and rapidly transforms generic hardware into a personalized computing environment. Both the accuracy and speed of transformation are important. The design of ISR pays careful attention to ease of deployment, a key requirement for universal infrastructure. Specifically, ISR sites can be set up and easily managed by relatively unskilled personnel.

ISR reduces dependence on mobile hardware. A user need only carry hardware larger than a dongle under two conditions: when traveling to destinations with poor network or hardware capability; or when using hardware-integrated applications such as augmented reality.

Seamless computation in spite of user movement is the holy grail of mobile computing. ISR is an important step toward this goal. Exploiting advance knowledge of travel, transferring VM state incrementally, and using transportable storage are all important techniques for making ISR viable. With these techniques, resume latency can be reduced to little more than the typical delay one experiences when opening a laptop. By leveraging the consistency of a distributed file system, ISR is robust in the face of a variety of human errors, and is tolerant of environments that are not well-managed. These attributes give us confidence that ISR can play an important role in the future of mobile computing.

References

[1] CHEN, P.M., NOBLE, B.D. When Virtual is Better Than Real. In *Proceedings of the 8th IEEE Workshop on Hot Topics in Operating Systems* (Schloss Elmau, Germany, May 2001).

[2] KOZUCH, M., SATYANARAYANAN, M. Internet Suspend/Resume. In *Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications* (Callicoon, NY, June 2002).

[3] RICHARDSON, T., STAFFORD-FRASER, Q., WOOD, K. R., AND HOPPER, A. Virtual Network Computing. *IEEE Internet Computing 2*, 1 (Jan/Feb 1998).

[4] SAPUNTZAKIS, C., CHANDRA, R., PFAFF, B., CHOW, J., LAM, M., ROSENBLUM, M. Optimizing the Migration of Virtual Computers. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation* (Boston, MA, Dec. 2002).

[5] SATYANARAYANAN, M. Scalable, Secure and Highly Available Distributed File Access. *IEEE Computer 23*, 5 (May 1990).

[6] SATYANARAYANAN, M. The Evolution of Coda. *ACM Transactions on Computer Systems 20*, 2 (May 2002).

[7] SCHMIDT, B.K., LAM, M.S., NORTHCUTT, J.D. The Interactive Performance of SLIM: a Stateless, Thin-Client Architecture. In *Proceedings of the 17th ACM Symposium on Operating Systems and Principles* (Kiawah Island, SC, December 1999).

[8] SMITH, S.W., AUSTEL, V. Trusting Trusted Hardware: Toward a Formal Model for Programmable Secure Coprocessors. In *Proceedings of the Third USENIX Workshop on Electronic Commerce* (Boston, MA, August 1998).

[9] SOLOMITA, E., KEMPF, J., DUCHAMP, D. XMOVE: A pseudoserver for X window movement. *The X Resource 11*, 1 (1994).

[10] TRUMAN, T.E., PERING, T., DOERING, R., BRODERSEN, R.W. The InfoPad Multimedia Terminal: A Portable Device for Wireless Information Access. *IEEE Transactions on Computers 47*, 10 (October 1998).

[11] TYGAR, J.D., YEE, B. Dyad: A System for Using Physically Secure Coprocessors. In *Proceedings of the Joint Harvard MIT Workshop on Technological Strategies for Protecting Intellectual Property in the Networked Multimedia Environment* (April 1993).

	Baseline	Fully Proactive	Pure Demand-Fetch	Dongle LKA
100 Mb/s	2504 (18)	10.3 (0.1)	14 (0.5)	13 (2.2)
10 Mb/s	5158 (34)	10.2 (0.0)	39 (0.4)	12 (0.5)
1 Mb/s	$>$ 9 hours	10.2 (0.0)	317 (0.3)	12 (0.3)
100 Kb/s	$>$ 90 hours	11.4 (0.0)	4301 (0.6)	12 (0.1)

This table shows resume latency (in seconds) for different state transfer policies at various bandwidths. In each case, the mean of three trials is reported, along with the standard deviation in parentheses. The results in the "Baseline" column for 1 Mb/s and 100 Kb/s are estimated rather than measured values.

Figure 4. Resume Latency

	Baseline	Fully Proactive	Pure Demand-Fetch	DVD LKA	Jukebox LKA
100 Mb/s	1105 (9)	*same as baseline*	1160 (5.6)	1141 (35.7)	1068 (5.7)
10 Mb/s	1170 (47)	*same as baseline*	1393 (20.1)	1186 (17.3)	1131 (6.8)
1 Mb/s	1272 (65)	*same as baseline*	4722 (69)	2128 (33.6)	2256 (24.7)
100 Kb/s	1409 (38)	*same as baseline*	42600 (918)	13967 (131.4)	13514 (30.7)

This table shows the running time (in seconds) of the CDA benchmark at different bandwidths for the state transfer policies examined in this paper. Each data point is the mean of three trials, with standard deviation in parentheses.

Figure 5. Elapsed Time for Benchmark